Posted by wolverin on Tue, 01 Aug 2023 06:57:11 GMT
View Forum Message <> Reply to Message

Posted by BlackEric on Tue, 01 Aug 2023 07:39:17 GMT
View Forum Message <> Reply to Message

How to get Windows DNS suffix search list using c++

Posted by wolverin on Tue, 01 Aug 2023 07:45:35 GMT
View Forum Message <> Reply to Message

Posted by BlackEric on Tue, 01 Aug 2023 08:15:01 GMT
View Forum Message <> Reply to Message

et\Services\Tcpip\Parameters

IpConfig. Win 10

Posted by wolverin on Tue, 01 Aug 2023 08:27:15 GMT
View Forum Message <> Reply to Message

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters\Interfaces

Posted by BlackEric on Tue, 01 Aug 2023 08:36:59 GMT
View Forum Message <> Reply to Message

Posted by wolverin on Tue, 01 Aug 2023 08:48:41 GMT
View Forum Message <> Reply to Message

https://learn.microsoft.com/ru-ru/windows/win32/api/iphlpapi/

Posted by SD on Tue, 01 Aug 2023 11:03:02 GMT
View Forum Message <> Reply to Message

https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi /nf-iphlpapi-getadaptersaddresses
https://learn.microsoft.com/en-us/windows/win32/api/netioapi
/nf-netioapi-getinterfacednssettings

Posted by BlackEric on Tue, 01 Aug 2023 12:05:40 GMT
View Forum Message <> Reply to Message

Posted by wolverin on Tue, 01 Aug 2023 17:50:34 GMT
View Forum Message <> Reply to Message

https:// learn.microsoft.com/ru-ru/windows/win32/api/iptypes/ns-iptyp
es-ip_adapter_addresses_lh?redirectedfrom=MSDN

Posted by wolverin on Wed, 02 Aug 2023 03:59:46 GMT
View Forum Message <> Reply to Message

https:// learn.microsoft.com/en-us/windows/win32/api/iptypes/ns-iptyp
es-ip_adapter_addresses_xp

https:// www.freepascal.org/daily/packages/winunits-jedi/jwaiptypes/_ ip_adapter_addresses.html

Posted by wolverin on Wed, 02 Aug 2023 04:46:33 GMT
View Forum Message <> Reply to Message

```
function GetHostDomain: string;
  const
    DLL = 'IPHLPAPI.DLL';

    MAX_ADAPTER_ADDRESS_LENGTH = 8;
    GAA_FLAG_INCLUDE_ALL_INTERFACES = $100;
    WORKING_BUFFER_SIZE = 15000;
    MAX_TRIES = 3;

  type
```

```
  ULONGLONG = UINT64;

  PSOCKADDR = ^SOCKADDR;
  SOCKADDR = record
    sa_family: u_short;
    sa_data: array[1..14] of Char;
  end;

  SOCKET_ADDRESS = record
    lpSockaddr: PSOCKADDR;
    iSockaddrLength: Integer;
  end;

  IP_PREFIX_ORIGIN = (
    IpPrefixOriginOther = 0,
    IpPrefixOriginManual,
    IpPrefixOriginWellKnown,
    IpPrefixOriginDhcp,
    IpPrefixOriginRouterAdvertisement,
    IpPrefixOriginUnchanged = $10
  );

  IP_SUFFIX_ORIGIN = (
    NlsoOther = 0,
    NlsoManual,
    NlsoWellKnown,
    NlsoDhcp,
    NlsoLinkLayerAddress,
    NlsoRandom,
    IpSuffixOriginOther = 0,
    IpSuffixOriginManual,
    IpSuffixOriginWellKnown,
    IpSuffixOriginDhcp,
    IpSuffixOriginLinkLayerAddress,
    IpSuffixOriginRandom,
    IpSuffixOriginUnchanged = $10
  );

  IP_DAD_STATE = (
    NldsInvalid,
    NldsTentative,
    NldsDuplicate,
    NldsDeprecated,
    NldsPreferred,
    IpDadStateInvalid = 0,
    IpDadStateTentative,
    IpDadStateDuplicate,
    IpDadStateDeprecated,
```

```
  IpDadStatePreferred
);

PIP_ADAPTER_UNICAST_ADDRESS = ^IP_ADAPTER_UNICAST_ADDRESS;
IP_ADAPTER_UNICAST_ADDRESS = record union: record
  case Integer of
    0: (Alignment: UInt64);
    1: (Length: ULONG;
        Flags: DWORD);
  end;
  Next: PIP_ADAPTER_UNICAST_ADDRESS;
  Address: SOCKET_ADDRESS;
  PrefixOrigin: IP_PREFIX_ORIGIN;
  SuffixOrigin: IP_SUFFIX_ORIGIN;
  DadState: IP_DAD_STATE;
  ValidLifetime: ULONG;
  PreferredLifetime: ULONG;
  LeaseLifetime: ULONG;
end;

PIP_ADAPTER_ANYCAST_ADDRESS = ^IP_ADAPTER_ANYCAST_ADDRESS;
IP_ADAPTER_ANYCAST_ADDRESS = record union: record
  case Integer of
    0: (Alignment: UINT64);
    1: (Length: ULONG;
        Flags: DWORD);
  end;
  Next: PIP_ADAPTER_ANYCAST_ADDRESS;
  Address: SOCKET_ADDRESS;
end;

PIP_ADAPTER_MULTICAST_ADDRESS = ^IP_ADAPTER_MULTICAST_ADDRESS;
IP_ADAPTER_MULTICAST_ADDRESS = record union: record
  case Integer of
    0: (Alignment: UINT64);
    1: (Length: ULONG;
        Flags: DWORD);
  end;
  Next: PIP_ADAPTER_MULTICAST_ADDRESS;
  Address: SOCKET_ADDRESS;
end;

PIP_ADAPTER_DNS_SERVER_ADDRESS = ^IP_ADAPTER_DNS_SERVER_ADDRESS;
IP_ADAPTER_DNS_SERVER_ADDRESS = record union: record
  case Integer of
    0: (Alignment: UINT64);
    1: (Length: ULONG;
        Reserved: DWORD);
```

```
  end;
  Next: PIP_ADAPTER_DNS_SERVER_ADDRESS;
  Address: SOCKET_ADDRESS;
end;

PIP_ADAPTER_PREFIX = ^IP_ADAPTER_PREFIX;
IP_ADAPTER_PREFIX = record union: record
  case Integer of
    0: (Alignment: UINT64);
    1: (Length: ULONG;
        Flags: DWORD);
  end;
  Next: PIP_ADAPTER_PREFIX;
  Address: SOCKET_ADDRESS;
  PrefixLength: ULONG;
end;

IF_OPER_STATUS = (
  IfOperStatusUp = 1,
  IfOperStatusDown,
  IfOperStatusTesting,
  IfOperStatusUnknown,
  IfOperStatusDormant,
  IfOperStatusNotPresent,
  IfOperStatusLowerLayerDown
);

PIP_ADAPTER_ADDRESSES = ^IP_ADAPTER_ADDRESSES;
IP_ADAPTER_ADDRESSES = record union: record
  case Integer of
    0: (Alignment: ULONGLONG);
    1: (Length: ULONG;
        IfIndex: DWORD);
  end;
  Next: PIP_ADAPTER_ADDRESSES;
  AdapterName: PCHAR;
  FirstUnicastAddress: PIP_ADAPTER_UNICAST_ADDRESS;
  FirstAnycastAddress: PIP_ADAPTER_ANYCAST_ADDRESS;
  FirstMulticastAddress: PIP_ADAPTER_MULTICAST_ADDRESS;
  FirstDnsServerAddress: PIP_ADAPTER_DNS_SERVER_ADDRESS;
  DnsSuffix: PWCHAR;
  Description: PWCHAR;
  FriendlyName: PWCHAR;
  PhysicalAddress: array[1..MAX_ADAPTER_ADDRESS_LENGTH] of BYTE;
  PhysicalAddressLength: DWORD;
  Flags: DWORD;
  Mtu: DWORD;
  IfType: DWORD;
```

```
      OperStatus: IF_OPER_STATUS;
      Ipv6IfIndex: DWORD;
      ZoneIndices: array[0..15] of DWORD;
      FirstPrefix: PIP_ADAPTER_PREFIX;
    end;

  var
    lib: THandle;
    GetAdaptersAddresses: function(Family: ULONG; Flags: ULONG; Reserved: Pointer;
AdapterAddresses: PIP_ADAPTER_ADDRESSES; SizePointer: PULONG): ULONG; stdcall;
    pAddr, pAddrCurr: PIP_ADAPTER_ADDRESSES;
    len: ULONG;
    i: Integer;
    rc: DWORD;

  Label
    _END;
begin
Result := '';

lib := LoadLibrary(DLL);
if lib = 0 then exit;

GetAdaptersAddresses := GetProcAddress(lib, PAnsiChar(AnsiString('GetAdaptersAddresses')));
if Not Assigned(GetAdaptersAddresses) then goto _END;

pAddr := nil;
len := WORKING_BUFFER_SIZE;
rc := NO_ERROR;
for i := 1 to MAX_TRIES do
    begin
    if Assigned(pAddr) then FreeMem(pAddr);
    GetMem(pAddr, len);
    if Not Assigned(pAddr) then goto _END;

    rc := GetAdaptersAddresses(AF_INET, GAA_FLAG_INCLUDE_ALL_INTERFACES, nil, pAddr,
@len);
    if rc <> ERROR_BUFFER_OVERFLOW then Break;
    end;

if rc = NO_ERROR then
   begin
   pAddrCurr := pAddr;
   while Assigned(pAddrCurr) do
        begin
        if Assigned(pAddrCurr.DnsSuffix) then
          if Length(pAddrCurr.DnsSuffix) > 0 then
             begin
```

```
            Result := pAddrCurr.DnsSuffix;
            Break;
            end;
        pAddrCurr := pAddrCurr.Next;
        end;
   end;

FreeMem(pAddr);

_END:
  FreeLibrary(lib);
end;
```